MCO1:

Machine Project Sorting

CCDSALG S15 GROUP 1

GUILLERMO, Eugene Siy

SIMBAHON, Joolz Ryane Chavez

VILLANUEVA, Nathaniel Frigillana

Ms. Ma. Christine Gendrano

**Algorithm Design**

* MERGE SORT

We first read the pseudocode for the merge sort algorithm. We then tried to modify the merge sort to take in two arrays of integers and strings. However, we realized that since the substrings get smaller by removing a suffix character to the far left. We then just used a single array of integer values to access the string at multiple points. This resulted in a more efficient code. Several revisions were made which resulted in the current algorithm below:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**MERGE (nArr[], cStr[], nLeft, nMid, nRight)**

nLenL = nMid - nLeft + 1;

nLenR = nRight - nMid;

let L[nLenL] and R[nLenR] be new arrays

Copy the left part of the suffix array into the L array

**for** i = 0 to nLenL

L[i] = Arr[nLeft + i];

Copy the right part of the suffix array into the R array

**for** j = 0 to nLenR

R[i] = Arr[nMid + j + 1];

i = j = 0;

k = nLeft;

**while** k to nRight and i to nLenL and j to nLenR

**if** strcmp(cStr + L[i] and cStr + R[j] <= 0)

Arr[k] = L[i];

i++;

**else**

Arr[k] = R[j];

j++;

k = k + 1;

**while** i < nLenL

Arr[k] = L[i];

k = k + 1;

i = i + 1;

**while** j < nLenR

Arr[k] = R[i];

k = k + 1;

j = j + 1;

**MERGE-SORT (nArr[], cStr[], p, r)**

**if** p < r

q = [(p + r)/2]

MERGE-SORT(Arr, cStr, p, q);

MERGE-SORT(Arr, cStr, q + 1, r)

MERGE(Arr, cStr, p, q, r);

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* INSERTION SORT

The pseudocode of insertion sort used an array that started at Index 1. For our implementation, we made the array start at Index 0 as array indices start from 0 in C language. In the original sorting algorithm, the shifting of elements relied on comparing the integer value at index i (A[i]) with an integer key.

Since we were tasked to create a suffix array, we needed to compare the suffixes lexicographically. In order to achieve this, the **strcmp** string function was used. In our first modification, we had an integer and string array, which were both sorted in our algorithm. However, later on, we realized that we could simply pass one string as a parameter and access its suffixes, using the starting points stored in the integer array. 

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**INSERTION-SORT (intA[], strA[])**

// nLen = strA.length

**for** j = 2 to nLen

strKey = strA + intA[j]

nKey = intA[j]

i = j – 1

**while** i >= 0 and strA + intA[j] > strKey

intA[i + 1] = intA[i]

i = i – 1

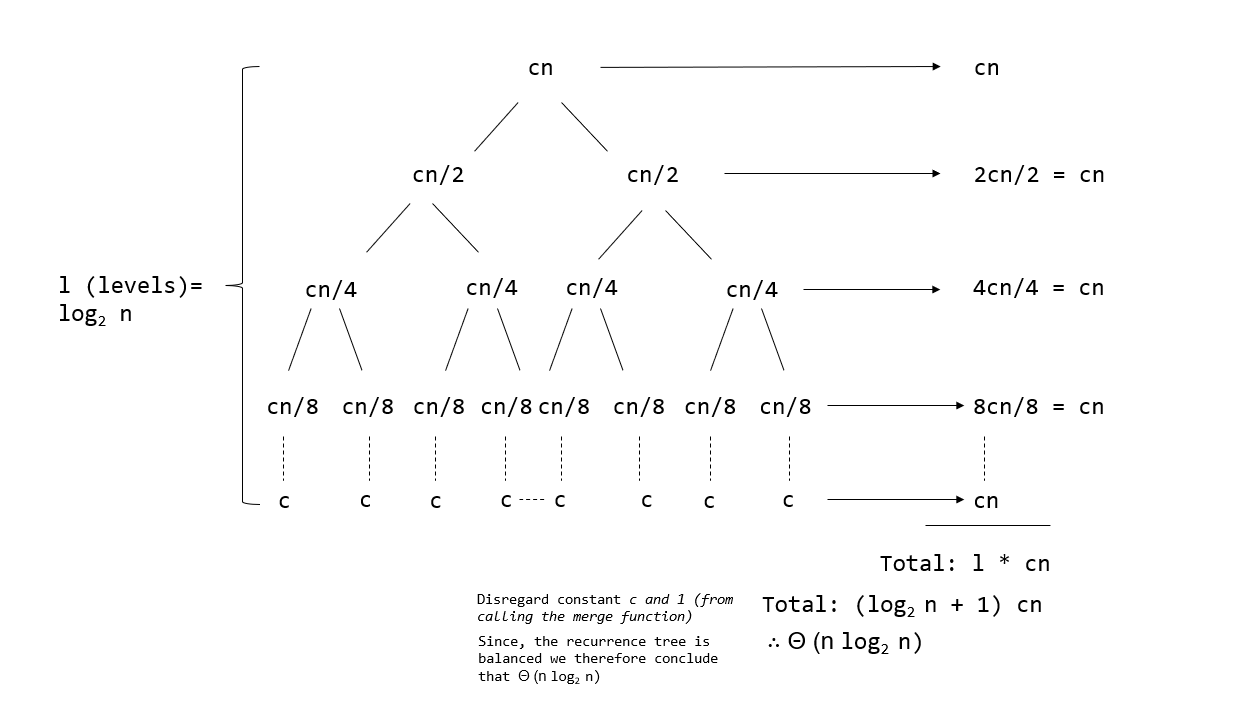
intA[i + 1] = nKey

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Theoretical Running Time Analysis**

* MERGE SORT

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

∴ Θ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* INSERTION SORT

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**INSERTION-SORT** **(intA[], strA[])** Let nLen = n

**for** j = 2 to nLen n - 2 + 2 = n

strKey = strA + intA[j] n - 2 + 1 = n - 1

nKey = intA[j] n - 1

i = j – 1 n - 1

**while** i >= 0 and strA + intA[j] > strKey n(n + 1)/2

intA[i + 1] = intA[i] n(n - 1)/2

i = i – 1 n(n - 1)/2

intA[i + 1] = nKey n - 1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

T(n) =

T(n) =

∴ Θ

**Theoretical Memory Usage Analysis**

* MERGE SORT

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**MERGE**

)

**MERGESORT**

)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* INSERTION SORT

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**INSERTION-SORT**

S(n) =

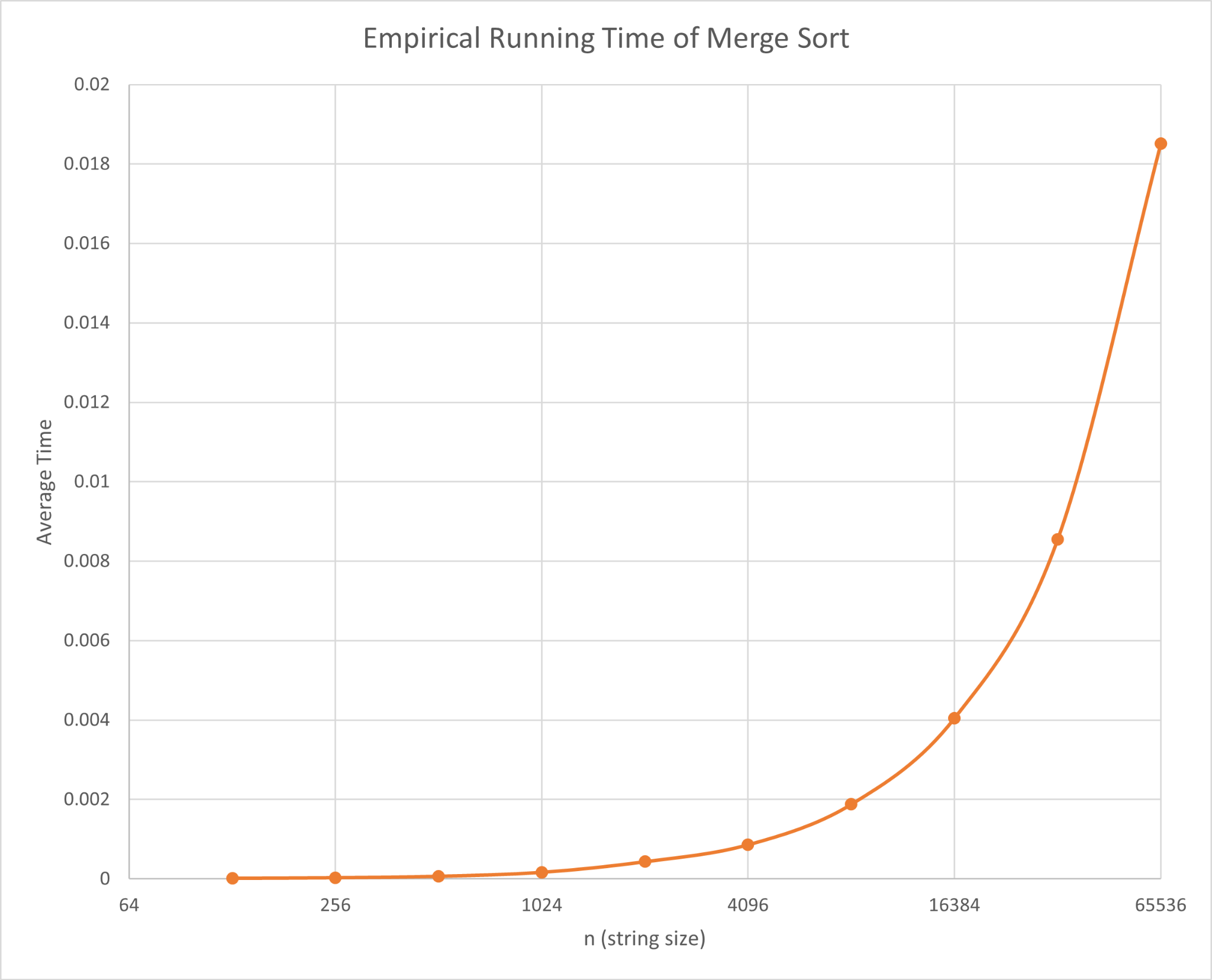
S(n) =

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Empirical Analysis of Running Time**

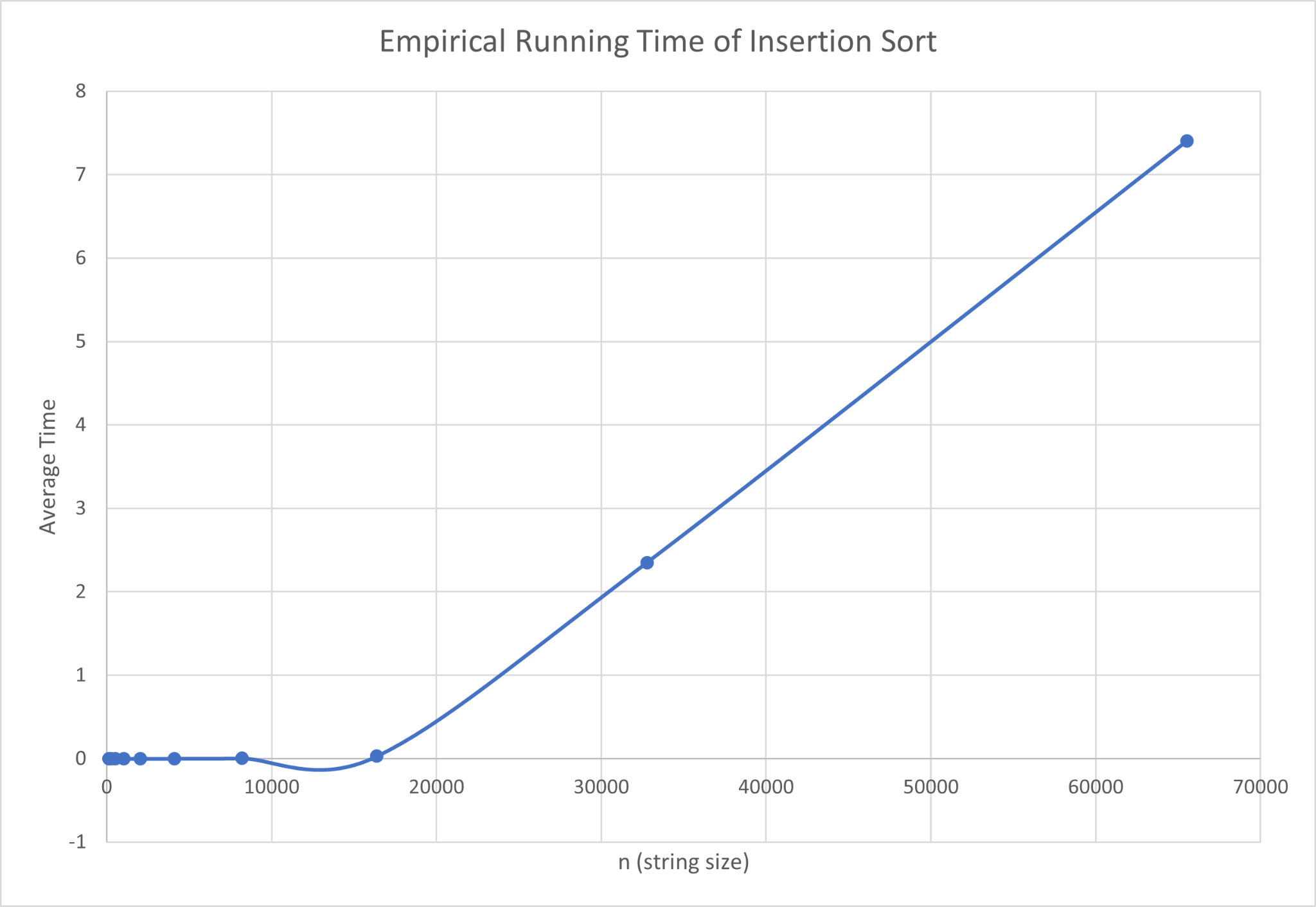
* MERGE SORT
  + k = 1000

| n | Average Time |
| --- | --- |
| 128 | 0.000018 s |
| 256 | 0.000032 s |
| 512 | 0.000069 s |
| 1024 | 0.000168 s |
| 2048 | 0.000435 s |
| 4096 | 0.000862 s |
| 8192 | 0.001881 s |
| 16384 | 0.004050 s |
| 32768 | 0.008550 s |
| 65536 | 0.018511 s |

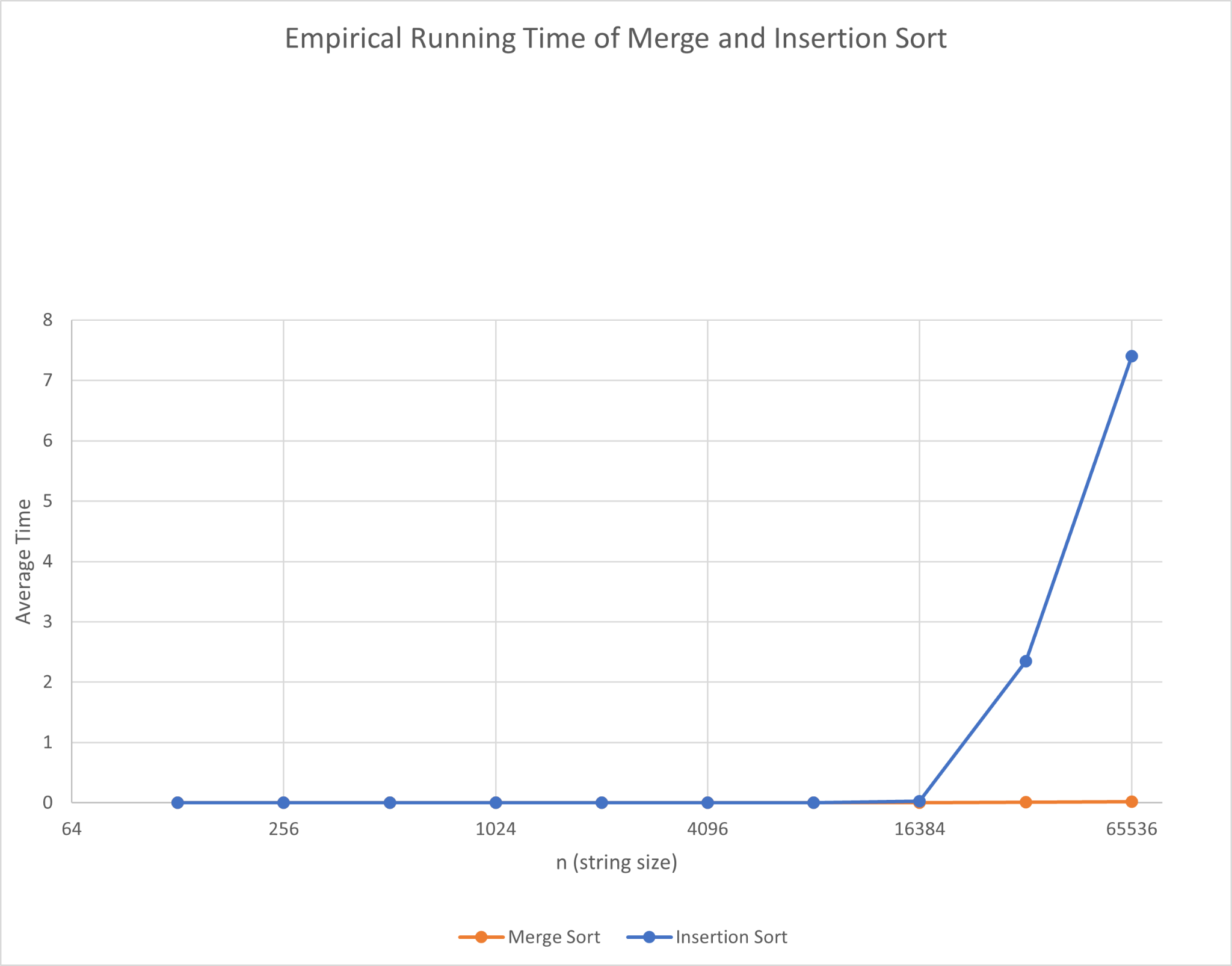


* INSERTION SORT
  + k = 1000

| n | Average Time |
| --- | --- |
| 128 | 0.000002 s |
| 256 | 0.000006 s |
| 512 | 0.000025 s |
| 1024 | 0.000108 s |
| 2048 | 0.000354 s |
| 4096 | 0.001246 s |
| 8192 | 0.004880 s |
| 16384 | 0.029268 s |
| 32768 | 2.350661 s |
| 65536 | 7.402814 s |



* INSERTION SORT VS MERGE SORT



**References**

*Algorithms | Computer science | Computing. (n.d.). Khan Academy. Retrieved October 25, 2022, from https://www.khanacademy.org/computing/computer-science/algorithms*

*Allocating string with malloc*. (2017, January 24). Stack Overflow. Retrieved October 21, 2022, from https://stackoverflow.com/questions/41830461/allocating-string-with-malloc

*C Program for Merge Sort*. (2022, June 3). GeeksforGeeks. Retrieved October 21, 2022, from https://www.geeksforgeeks.org/c-program-for-merge-sort/

*Execution time of C program*. (2011, March 9). Stack Overflow. Retrieved October 22, 2022, from https://stackoverflow.com/questions/5248915/execution-time-of-c-program

Gendrano, C. (2022, September 23). *CCDSALG Ch 02 - Sorting*. Google Slides. Retrieved October 21, 2022, from https://docs.google.com/presentation/d/1iwJ1H65gI4jn\_07HFdjRSOVA5bWDCnTnDKc9mlW\_tFs/edit#slide=id.p

*How to generate a random int in C?* (2009, May 4). Stack Overflow. Retrieved October 22, 2022, from https://stackoverflow.com/questions/822323/how-to-generate-a-random-int-in-c

Lecture 20: Recursion Trees and the Master Method. (n.d.). Retrieved October 25, 2022, from https://www.cs.cornell.edu/courses/cs3110/2012sp/lectures/lec20-master/lec20.html

Running time of binary search (article). (n.d.). Khan Academy. Retrieved October 25, 2022, from https://www.khanacademy.org/computing/computer-science/algorithms/binary-search/a/running-time-of-binary-search

*Segmentation Fault, large arrays*. (2011, October 26). Stack Overflow. Retrieved October 21, 2022, from https://stackoverflow.com/questions/7902228/segmentation-fault-large-arrays

**Description of the Work Distribution**

We first made code for both insertion and merge sort. Eugene and Nathaniel compared their code on merge sort while Joolz made insertion sort. We then checked each other’s code and revised them little by little. Joolz and Nathaniel computed the theoretical time and memory usage. Eugene made an additional function for randomizing strings and made test cases in the main function. We then tried to compile and run the code. Nathaniel then compiled the averages and plotted them into graphs. We finally read through and proofread the whole documentation.